

# Simplification of Training Data for Cross-Project Defect Prediction

Peng He<sup>a,b</sup>, Bing Li<sup>c,d</sup>, Deguang Zhang<sup>a,b</sup>, Yutao Ma<sup>b,d,\*</sup>

<sup>a</sup>State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China

<sup>b</sup>School of Computer, Wuhan University, Wuhan 430072, China

<sup>c</sup>International School of Software, Wuhan University, Wuhan 430079, China

<sup>d</sup>Research Center for Complex Network, Wuhan University, Wuhan 430072, China

## Abstract

Cross-project defect prediction (CPDP) plays an important role in estimating the most likely defect-prone software components, especially for new or inactive projects. To the best of our knowledge, few prior studies provide explicit guidelines on how to select suitable training data of quality from a large number of public software repositories. In this paper, we have proposed a training data simplification method for practical CPDP in consideration of multiple levels of granularity and filtering strategies for data sets. In addition, we have also provided quantitative evidence on the selection of a suitable filter in terms of defect-proneness ratio. Based on an empirical study on 34 releases of 10 open-source projects, we have elaborately compared the prediction performance of different defect predictors built with five well-known classifiers using training data simplified at different levels of granularity and with two popular filters. The results indicate that when using the multi-granularity simplification method with an appropriate filter, the prediction models based on Naïve Bayes can achieve fairly good performance and outperform the benchmark method.

**Keywords:** cross-project defect prediction, training data simplification, software quality, data mining, transfer learning

## 1. Introduction

Software defect prediction is a research field that seeks effective methods for predicting the defect-proneness in a given software component. These methods can help software engineers allocate limited resources to those components that are most likely to contain defects in testing and maintenance activities. Early studies in this field usually focused on Within-Project Defect Prediction (WPDP), which trained defect predictors from the data of historical releases in the same project and predicted defects in the upcoming releases or reported the results of cross-validation on the same data set (He et al., 2012). Zimmermann et al. (2009) stated that defect prediction performs well within projects as long as there is a sufficient amount of data available to train prediction models. However, such an assumption does not always hold in practice, especially for newly-created or inactive software projects. For example, Rainer et al. (2005) conducted an in-depth analysis on SourceForge<sup>1</sup> and found that only 1% of software projects on SourceForge were actually active in terms of their metrics.

Fortunately, there are many on-line public defect data sets from other projects that are freely available and can be used as training data sets (TDSs), such as PROMISE<sup>2</sup> and Apache<sup>3</sup>. Thus, some researchers have been inspired to overcome the above problem of WPDP by means of Cross-Project Defect Prediction (CPDP) (He et al., 2012; Zimmermann et al., 2009; Peters et al., 2013; Rahman et al., 2012; Briand et al., 2002;

Turhan et al., 2009; Herbold, 2013). In general, CPDP is the art of using the data from other projects to predict software defects in the target project with a very small amount of local data. CPDP models have been proven to be feasible by many previous studies (He et al., 2012; Rahman et al., 2012). However, He et al. (2012) found that the overall performance of CPDP was drastically improved with suitable training data, while Turhan et al. (2009) also affirmed that using a complete TDS would lead to excessive false alarms. That is, data quality, rather than the total quantity of data, is more likely to affect the outcomes of CPDP to some extent.

There is no doubt that the availability of defect data sets on the Internet will continue to grow, as will the popularity of open-source software. The construction of an appropriate TDS of quality gathered from a large number of public software repositories is still a challenge for CPDP (Herbold, 2013). To the best of our knowledge, there are two primary ways to investigate this issue. On the one hand, many researchers have attempted to reduce data dimensions using feature selection techniques, and numerous studies have validated that a reduced feature subset can improve the performance and efficiency of defect prediction (Lu et al., 2012; He et al., 2014). On the other hand, few researchers have attempted to simplify a TDS by reducing the volume of data (He et al., 2012; Peters et al., 2013) to exclude irrelevant training data and retain those that are most suitable.

Figure 1 shows a simple summary of the state-of-the-art methods related to the topics of interest in this paper (see the contents with a gray background). Prior studies have attempted to reduce irrelevant training data at different levels of granularity, e.g., release-level (He et al., 2012) and instance/file-level (Turhan et al., 2009). Unfortunately, they all dealt with training data simplification based on a single

\*Corresponding author. Tel: +86 27 68776081

E-mail: {penghe (P. He), bingli (B. Li), deguangzhang (D.G. Zhang), ytma (Y.T. Ma)}@whu.edu.cn

<sup>1</sup><http://sourceforge.net>

<sup>2</sup><http://promisedata.org>

<sup>3</sup><http://www.apache.org>

level of granularity. Furthermore, different filtering strategies were recently proposed to improve the selection of suitable training instances in a TDS (Peters et al., 2013). Although these methods seem very promising separately, we actually do not know how to choose the most appropriate filter when dealing with a specific defect data set of a given project. In other words, they did not offer any practical guidelines for the decision-making on which granularity, strategy for instance selection and classifier should be preferably selected in a specific scenario.

Considering the importance of defect prediction in software development and maintenance phases, TDS simplification on data volume is the key to achieving better prediction results, as the data from other projects available on the Internet is ever-increasing. As shown in Figure 1, to obtain an appropriate TDS of quality, we should take the two chief factors affecting training data simplification into account. Hence, the goal of this study is to propose a method to simplify a large amount of training data for CPDP in terms of different levels of granularity and filtering strategies for instance selection. We also attempt to discover useful guiding principles that can assist software engineers in building suitable defect predictors. To accomplish the above goals, we focus mainly on exploring the following research questions:

*RQ1: Does our TDS simplification method perform well compared with the benchmark methods?*

The quality of training data is one of the important factors that determine the performance of a defect predictor. TDS simplification is performed to obtain high quality training data by removing irrelevant and redundant instances. The state-of-the-art simplification methods are designed at a single level of granularity of data, and each one has its good and bad points. Hence, the goal of this research question is to examine whether our method based on a multi-granularity simplification strategy performs as well as (or outperforms) those up-to-date methods.

*RQ2: Which classifier is more suitable for CPDP with our TDS simplification method?*

The findings of previous studies indicate that some simple classifiers perform well for CPDP without training data simplification, such as Logistic Regression and Naïve Bayes (Hall et al., 2012). For this research question, we would like to validate whether simple classifiers can also achieve better prediction results based on a simplified TDS.

*RQ3: Which filter for instance selection should be preferable in a specific scenario?*

The filtering strategy (also known as the filter) determines how those appropriate instances in a TDS are selected and preserved. Currently, two types of filters for instance selection exist, i.e., training set-driven filter and test set-driven filter. However, the application contexts of the two filters remain unclear. Thus, the goal of this research question is to find a quantitative rule for filter selection, to improve the prediction performance based on a single type of filter.

The contribution of our work is twofold:

- We proposed a multi-granularity TDS simplification method to obtain training data of quality for CPDP. Empirical results show that our method can filter out more irrelevant and redundant data compared with the benchmark method. Moreover, the predictors trained by the simplified TDS according to the method can achieve better prediction precision as a whole.
- We first provided practical decision rules for an appropriate choice between the two existing filtering strategies for training data simplification in terms of defect-proneness ratio. Empirical results show that the reasonable selection of filters can lead to better prediction performance than a single type of filter.

We believe that the results of our study could be a stepping stone for current and future approaches to practical CPDP, as well as a new attempt for software engineering data simplification with new learning techniques such as transfer learning in the era of Big Data.

The rest of this paper is organized as follows. Section 2 is a review of related work. In Section 3, we introduce the method for TDS simplification in detail, and in Section 4, we evaluate our experiments with a case study based on 10 open-source projects. Section 5 and Section 6 present and discuss our findings and the threats to validity, respectively. Finally, we conclude this paper and present an agenda for future work in Section 7.

## 2. Related Work

### 2.1. Cross-Project Defect Prediction

Because it is sometimes difficult for WPDP to collect sufficient historical data, CPDP is currently popular within the field of defect prediction. To the best of our knowledge, Briand et al. (2002) conducted the earliest study on CPDP, and they applied the prediction model built on Xpose to Jwriter. The authors validated that such a model performed better than the random model and outperformed it in terms of class size. However, Zimmermann et al. (2009) conducted a large-scale experiment on data vs. domain vs. process, and found that only 3.4% of 622 cross-project predictions actually worked. Interestingly, CPDP was not symmetrical between Firefox and Microsoft IE, that is, Firefox is a sound defect predictor for Microsoft IE, but not vice versa. Similar results are reported in (Menzies et al., 2013; Posnett et al., 2011; Bettenburg et al., 2012).

Turhan et al. (2009) proposed a nearest-neighbor filtering technique to prune away irrelevant cross-project data, and they analyzed the performance of CPDP based on 10 projects collected from the PROMISE repository. Moreover, they investigated the case where prediction models were constructed from a blend of within- and cross-project data, and concluded that in case there was limited local data (e.g., 10% of historical data) of a target project, such mixed project predictions were

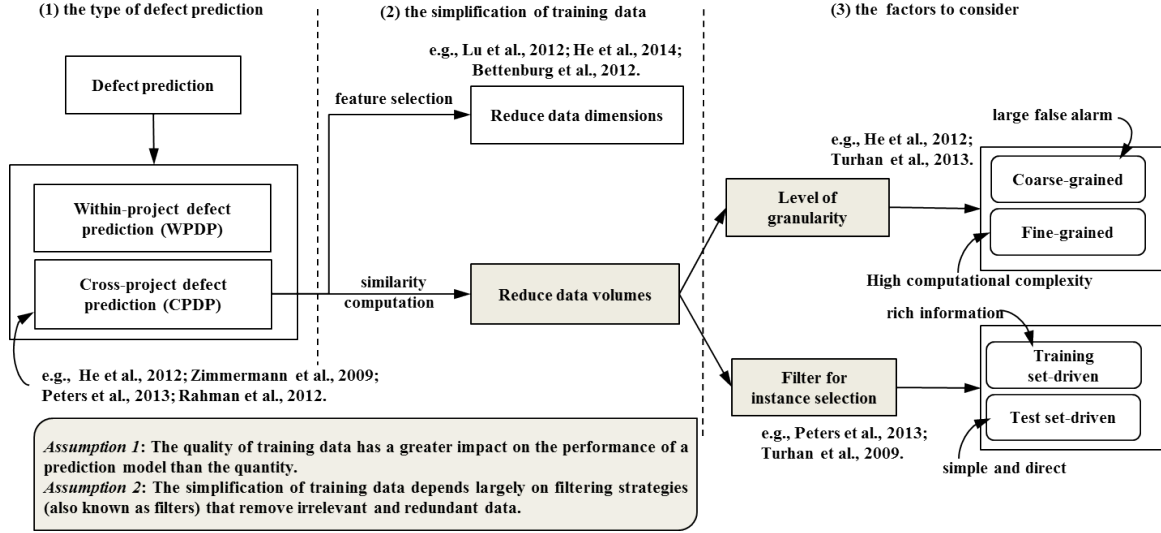


Figure 1: A summary of the state-of-the-art CPDP from the perspective of training data simplification.

viable, as they performed as well as within-project prediction models (Turhan et al., 2013).

Rahman et al. (2012) conducted a cost-sensitive analysis on the efficacy of CPDP based on 38 releases of nine large Apache Software Foundation (ASF) projects. Their findings revealed that the cost-sensitive cross-project prediction performance was not worse than the within-project prediction performance, and it was substantially better than the random prediction performance. Peters et al. (2013) introduced a new filter to realize better cross-company learning compared with the state-of-the-art Burak filter (Turhan et al., 2009). The results showed that their approach could build 64% more useful predictors than both within-company and cross-company approaches based on the Burak filter, and demonstrated that the training set-driven filter was able to achieve better prediction results for those projects without sufficient local data.

He et al. (2012) conducted three experiments on the same data sets used in this study to test and verify the idea that training data from other projects can provide acceptable prediction results. They further proposed an approach to automatically select suitable training data for those projects that lack local historical data. Towards efficient training data selection for CPDP, Herbold (2013) proposed several useful strategies according to 44 data sets from 14 open-source projects. The results demonstrated that their selection strategies improved the achieved success rate of CPDP significantly, but the quality of the results was still unable to outstrip WPDP.

The review reveals that previous studies focused mainly on the feasibility of CPDP and the selection of suitable training data at a single level of granularity of data. However, relatively little attention has been paid to empirically exploring the impact of TDS simplification in terms of different levels of granularity on prediction performance. Moreover, little is known about the decision rule for a proper choice among the existing filters for instance selection.

## 2.2. Defect Prediction with Transfer Learning

Transfer learning techniques have attracted more and more attention in machine learning and data mining over the last several years (Pan and Yang, 2010), and the successful applications include software effort estimation (Kocagune et al., 2014), text classification (Xue et al., 2008), name-entity recognition (Arnold et al., 2007), natural language processing (Pan et al., 2010) and email spam filtering (Zhang et al., 2007). Recently, CPDP was also deemed as a transfer learning problem. The problem setting of CPDP is related to the adaptation setting in transfer learning for building a classifier in the target project using the training data from those relevant source projects. Thus far, transfer learning techniques have been proven to be appropriate for CPDP in practice (Nam et al., 2013).

To harness cross-company defect datasets, Ma et al. (2012) utilized the transfer learning method to build faster and highly effective prediction models. They proposed a novel algorithm that used the information of all the suitable features in training data, known as Transfer Naïve Bayes (TNB), and the experimental result indicated that TNB was more accurate in terms of AUC (the area under the receiver operating characteristic curve) and less time-consuming than benchmark methods.

Nam et al. (2013) applied the transfer learning method, called TCA (Transfer Component Analysis), to find a latent feature space for the data of both training and test projects by minimizing the distance between the data distributions while preserving the original data properties. After learning the latent space in terms of six statistical characteristics, i.e., *mean*, *median*, *min*, *max*, *standard deviation* and the number of instances, the data of training and test projects will be mapped onto it to reduce the difference in the data distributions. The experimental results for eight open-source projects indicated that their method significantly improved CPDP performance.

In general, although the above studies improve the

performance of CPDP, they are time-consuming in that their experiments were conducted at the level of instances (files). In this study, to overcome the data distribution difference between source and target projects, we have also adopted the transfer learning method, which was applied to the releases available from different projects.

### 3. Methodology

In this paper, CPDP is defined as follows: Given a source project  $P_S$  and a target project  $P_T$ , CPDP aims to achieve the target prediction in  $P_T$  using the knowledge extracted from  $P_S$ , where  $P_T \neq P_S$ . Assuming that source and target projects have the same set of features, they may differ in feature distribution characteristics. The goal of our method is to learn a model from the selected source projects (training data) and apply the learned model to a target project (test data). Based on prior studies on CPDP, the TDS simplification process for CPDP is both explained in the following paragraphs and illustrated in Figure 2. Specifically, unlike previous studies, we introduce two levels of granularity and two types of filtering strategies for TDS simplification based on characteristic and instance vectors.

In brief, our method for TDS simplification has two key steps. The first step is selecting  $k$  candidate releases that are most similar to the target release in terms of data distributional characteristics. The second is choosing the  $k$  nearest instances of each test instance from those candidate releases according to suitable filtering strategies. Based on different classifiers, defect predictors can be trained from the simplified TDS, and then are applied to test data.

In our context, a release  $R$  contains  $m$  instances (.java files), represented as  $R = \{I_1, I_2, \dots, I_m\}$ . An instance can be represented as  $I_i = \{f_{i1}, f_{i2}, \dots, f_{in}\}$ , where  $f_{ij}$  is the  $j^{th}$  feature value of the instance  $I_i$ , and  $n$  is the number of features. Meanwhile, a feature vector can be represented as  $F_i = \{f_{1i}, f_{2i}, \dots, f_{mi}\}$ , where  $f_{ji}$  is the value of the  $j^{th}$  instance for the feature  $F_i$ , and  $m$  is the number of instances. An initial TDS—an aggregate of multiple data sets—is often comprised of many releases from different projects:  $S = \{R_1, R_2, \dots, R_l\}$ , where  $l$  is the number of releases. The distributional characteristic vector of a release can be formulated as  $V = \{C_1, C_2, \dots, C_k, \dots, C_n\}$ , where  $C_k$  is the distribution of the feature  $F_k$  and can be written as  $C_k = \{SC_1, SC_2, \dots, SC_s\}$  (see Figure 3). For the meaning of the statistical characteristics  $SC_s$ , please refer to Table 1.

#### 3.1. Level of Granularity

For CPDP, one of the easiest methods is to directly train prediction models without any TDS simplification methods. During this learning process, all of the data from other projects are utilized as a TDS. Take the experimental datasets used in this paper as an example; Table 2 shows the prediction results of CPDP without TDS simplification. Clearly, the average number of training instances is much greater than the size of each test set. More detailed information of the experimental datasets will be introduced in Section 4.1. In fact, our experimental

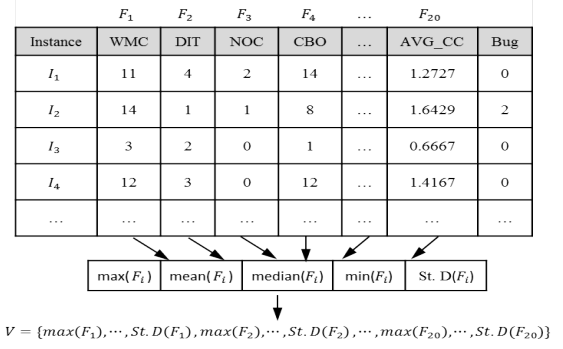


Figure 3: The structure of a release ( $R$ ) (instances ( $I$ ), features ( $F$ ) and distributional characteristics ( $V$ )): an example.

Table 1: Description of the indicators used to describe the distributional characteristics of a release

Indicator	Description
Median	The numeric value separating the higher half of a population from the lower half
Mean	The average value of samples in a population; specifically, it refers to arithmetic mean in this paper
Min	The least value in a population
Max	The greatest value in a population
St. D	The square root of the variance

data occupied a very small fraction of the public defect data available on the Internet. On the one hand, although it does not matter for computing resources and time complexity, for a learning process based on a vast amount of training data, it is cost-sensitive and not practical for software engineers; on the other hand, this decreases the accuracy of prediction models to some extent (He et al., 2012; Turhan et al., 2009). Therefore, how to obtain the right training data by TDS simplification becomes meaningful (Peters et al., 2013).

**rTDS:** The TDS simplification at the release level is a simple and coarse-grained method, referred to as *rTDS*. The coarse-grained simplification of training data often uses the k-Nearest Neighbors algorithm to measure the similarity (via Euclidean distance<sup>4</sup>) between the release  $V_{training}$  and the release  $V_{target}$ . That is, the  $k$  nearest candidate releases are selected as the ultimate TDS (He et al., 2012; Turhan et al., 2009; Herbold, 2013). In our study, a data set is a release of a project, and five commonly-used indicators, i.e., *max*, *min*, *median*, *mean*, and *standard deviation*, are involved in describing the statistical characteristics (SCs) of a release (see Table 1). Thus, the distance between two releases can be formulated as:  $distance_R = \sqrt{(SC_{i1} - SC_{j1})^2 + \dots + (SC_{is} - SC_{js})^2}$ .

**iTDS:** Compared with the *rTDS*, the fine-grained TDS simplification should be conducted based on the computation of the similarity between the instance  $I_{training}$  and the instance  $I_{target}$ , which is referred to as *iTDS*. It returns the  $k$  nearest training instances for each target instance  $I_{target}$  by calculating their Euclidean distance (Peters et al., 2013; Nam et al., 2013).

<sup>4</sup>[http://en.wikipedia.org/wiki/Euclidean\\_distance](http://en.wikipedia.org/wiki/Euclidean_distance)

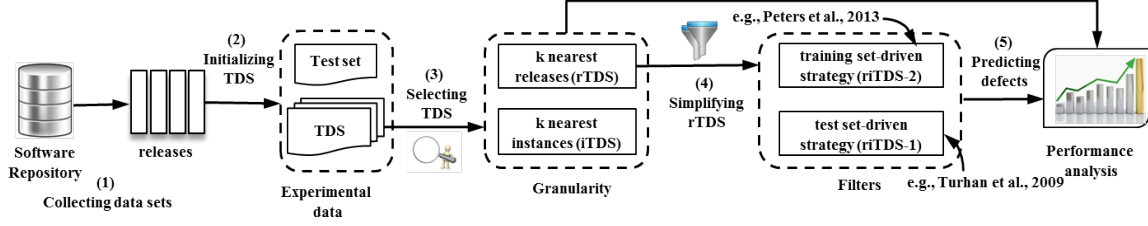


Figure 2: The process of TDS simplification for CPDP.

Table 2: The results of CPDP without TDS simplification. Numeric values in the second and third columns indicate the mean values of the measures. # instances (TDS) represents the average number of training instances in all TDSs in question.

Classifiers	f-measure	g-measure	# instances (TDS)
J48	0.369	0.499	11824
LR	0.291	0.358	
NB	0.464	0.617	
RF	0.322	0.432	
SVM	0.311	0.392	

Thus, the distance between two instances can be formulated as:  $distance_I = \sqrt{(f_{i1} - f_{j1})^2 + \dots + (f_{in} - f_{jn})^2}$ .

**riTDS:** Considering that there are a large number of on-line public defect data sets available for use as candidate training data, and that the number is still growing fast, it is impractical to completely calculate the distances of all instance pairs by the *iTDS*. However, using the *rTDS* alone may cause excessive false alarms because of the inclusion of many irrelevant training instances. Thus, we propose a two-step strategy for TDS simplification—*riTDS*, which obtains the coarse-grained set *rTDS* first and then simplifies it by a fine-grained method such as the *iTDS*. This strategy can be interpreted as a combination of the aforementioned two cases, also named as a multi-granularity simplification strategy. In other words, we first select the  $k$  nearest releases, instead of all releases available, as the candidate training data *rTDS*. Subsequently, we further simplify the coarse-grained set *rTDS* at the instance level according to suitable filters.

### 3.2. Filter for Instance Selection

For the *riTDS*, in the second step, there are two state-of-the-art filters for instance selection according to the choice of reference data. One is driven by the test set and returns the  $k$  nearest instances in the set *rTDS* for each test instance directly (abbreviated to *riTDS-1* in our context). This filter is to ensure that the information of each test instance is fully utilized, and it is referred to as a test set-driven filter. The other is just the opposite; it is training data-driven via labeling of the  $k$  nearest test instances for each training instance first and then returning of the nearest training instance of each labeled test instance (abbreviated to *riTDS-2* in our context). Clearly, in this case, it is possible that some test instances are never labeled as the nearest instance for certain training instances. Therefore, not all test instances will be utilized in favor of training instances.

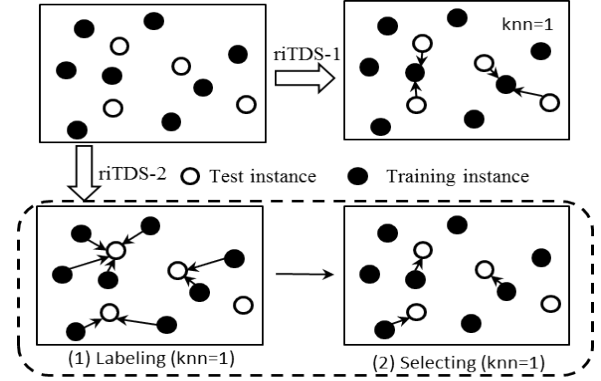


Figure 4: The description of two types of filters for instance selection.

The informal description of these two types of filters is shown in Figure 4. To the best of our knowledge, the Burak filter (Turhan et al., 2009) and the Peters filter (Peters et al., 2013) are the typical representatives of these two types of filters. For more details of their implementation, please refer to the related literature. Note that our primary goal in this section is to find some helpful guidelines for software engineers to definitely discriminate the application contexts of each filter, instead of improving the performance of these existing filters. Algorithm 1 formalizes the implementation of the *riTDS* with regard to these two filters.

## 4. Case Study

### 4.1. Data Setup

In this study, 34 releases of 10 open-source projects available in the PROMISE repository are used for our experiments. Detailed information of the releases is listed in Table 3, where #Instances represents the number of instances in a release, and the number of defects and the proportion of buggy instances are listed in the corresponding columns #Defects and %Defects, respectively. Each instance in a release represents a class (.java) file and consists of 20 software metrics (independent variables) and a binary label for the defect proneness (dependent variable). Table 4 presents all metrics used in this study as well as their descriptions. For those readers who are interested in the datasets, please refer to (Jureczko and Madeyski, 2010).

Before performing a cross-project defect prediction, we need to select a target data set and its appropriate TDS. Each one in the 34 releases was selected to be the target data set once, i.e.,

we repeated our approach for 34 different cross-project defect predictions. With regard to our primary objective, we set up an initial TDS for CPDP, which excluded any releases from the target project. For instance, for Xalan-2.5, the releases Xalan-2.4 and Xalan-2.6 cannot be included in its initial TDS.

---

**Algorithm 1** A two-step strategy for TDS simplification

---

**Input:**

- 1: Candidate TDS set  $S = \{R_1, R_2, \dots, R_N\}$ ;
- 2: Target release  $R_{target} = \{I_1, I_2, \dots, I_m\}$ ;
- 3: Number of selected releases  $r$ ;
- 4: Filtering strategy  $F = \{\text{training set-driven}, \text{test set-driven}\}$ ;

**Method:**

- 5: Let  $rTDS$  be the top  $r$  nearest releases of  $R_{target}$  in  $S$ ;
  - 6: Let  $riTDS$  be the simplified training set;
  - 7: Initialize  $rTDS \leftarrow \emptyset, riTDS \leftarrow \emptyset, r = 3$ ;
  - 8: **while**  $r > 0$  **do**
  - 9:    //  $r = 1, 2, 3$  in this paper
  - 10:    // return the  $r$  nearest releases for  $R_{target}$  in terms of  $distance_R$
  - 11:     $rTDS \leftarrow KNN(S, R_{target}, r)$ ;
  - 12:    **if**  $F \leftarrow \text{training set-driven}$  **then**
  - 13:      **for** each instance  $I \in R_{target}$  **do**
  - 14:        // return its  $k$  nearest instances in  $rTDS$  in terms of  $distance_I$
  - 15:         $tempSet \leftarrow KNN(rTDS, I, k)$ ;
  - 16:      **end for**
  - 17:       $riTDS \leftarrow tempSet$ ;
  - 18:    **else**
  - 19:      **for** each instance  $I \in rTDS$  **do**
  - 20:        // label its  $k$  nearest instances in  $R_{target}$  in terms of  $distance_I$
  - 21:         $labelMap \leftarrow Label(I, R_{target}, k)$ ;
  - 22:         $tempSet \leftarrow$  the set of labeled target instances;
  - 23:      **end for**
  - 24:      **for** each instance  $I \in tempSet$  **do**
  - 25:        // return its nearest instance  $I'$  ( $I' \in rTDS$ ) according to the  $labelMap$ , if a test instance's nearest instance has been chosen, select the next nearest one.
  - 26:         $riTDS \leftarrow riTDS \cup \{I'\}$ ;
  - 27:      **end for**
  - 28:    **end if**
  - 29:     $r --$ ;
  - 30: **end while**
  - 31: **return**  $riTDS$ ;
- 

Note that, there is a preprocessing that transforms the bug attribute into a binary value before using it as the dependent variable in our context. According to our prior work (He et al., 2014), we find that the majority of class files in the 34 data sets have no more than 3 defects, and the ratio of instances with more than 10 defects to the total instances is less than 0.2%. In a word, a class is non-buggy only if the number of bugs in it is equal to 0. Otherwise, it is buggy regardless of the number of bugs. Similar preprocessing has been used in several prior studies, such as (He et al., 2012; Peters et al., 2013; Turhan et al., 2009, 2013; Herbold, 2013).

Moreover, some prior studies have suggested that a logarithmic filter on numeric values might improve prediction performance because of the highly skewed distribution of

feature values (Turhan et al., 2009; Menzies et al., 2002). In this paper, for each numeric value  $f_{ij}$ ,  $f'_{ij} = \ln(f_{ij} + 1)$ , where  $f'_{ij}$  is the new value of the original value  $f_{ij}$ . There are some other commonly used methods for numeric values preprocessing, such as max-min and z-score methods (Nam et al., 2013).

Table 3: Details of the 34 data sets, including the number of instances (files) and defects and the defect rate.

No.	Releases	#Instances	#Defects	%Defects
1	Ant-1.3	125	20	16.0
2	Ant-1.4	178	40	22.5
3	Ant-1.5	293	32	10.9
4	Ant-1.6	351	92	26.2
5	Ant-1.7	745	166	22.3
6	Camel-1.0	339	13	3.8
7	Camel-1.2	608	216	35.5
8	Camel-1.4	872	145	16.6
9	Camel-1.6	965	188	19.5
10	Ivy-1.1	111	63	56.8
11	Ivy-1.4	241	16	6.6
12	Ivy-2.0	352	40	11.4
13	Jedit-3.2	272	90	33.1
14	Jedit-4.0	306	75	24.5
15	Lucene-2.0	195	91	46.7
16	Lucene-2.2	247	144	58.3
17	Lucene-2.4	340	203	59.7
18	Poi-1.5	237	141	59.5
19	Poi-2.0	314	37	11.8
20	Poi-2.5	385	248	64.4
21	Poi-3.0	442	281	63.6
22	Synapse-1.0	157	16	10.2
23	Synapse-1.1	222	60	27.0
24	Synapse-1.2	256	86	33.6
25	Velocity-1.4	196	147	75.0
26	Velocity-1.5	214	142	66.4
27	Velocity-1.6	229	78	34.1
28	Xalan-2.4	723	110	15.2
29	Xalan-2.5	803	387	48.2
30	Xalan-2.6	885	411	46.4
31	Xerces-init	162	77	47.5
32	Xerces-1.2	440	71	16.1
33	Xerces-1.3	453	69	15.2
34	Xerces-1.4	588	437	74.3

#### 4.2. Experimental Design

Based on the prediction results of the predictors trained without TDS simplification (see Table 2), the entire framework of our experiments is illustrated in Figure 5.

First, to make a comparison between our method and the benchmark methods, three types of TDS simplification methods were considered in our experiments: (1) coarse-grained TDS simplification ( $rTDS$ ), which uses the nearest  $k$  training releases of the target release as training data; (2) fine-grained TDS simplification ( $iTDS$ ), which uses the nearest  $k$  training instances of each target instance as training data; and (3) multi-granularity TDS simplification ( $riTDS$ ), which selects suitable training instances from the set  $rTDS$ . For the  $rTDS$  and

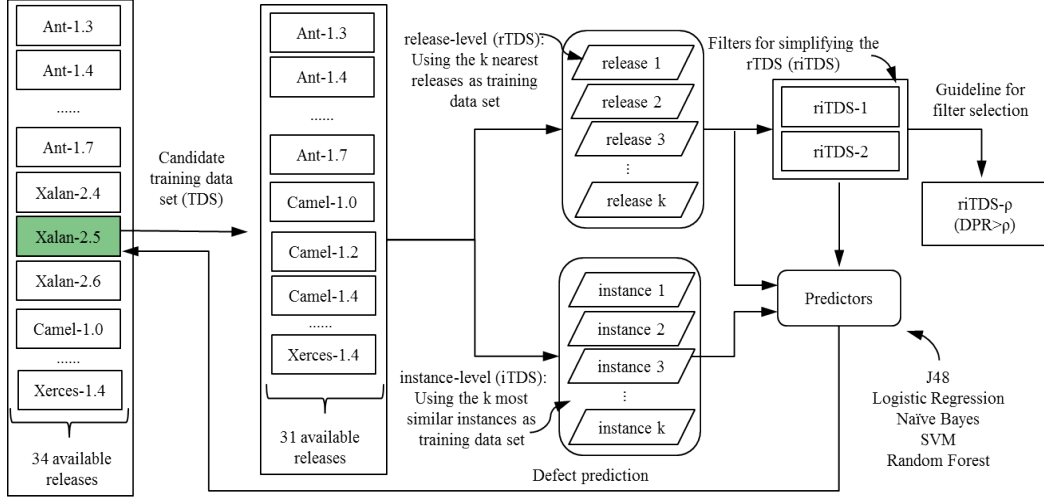


Figure 5: The framework of our approach—an example of the target project Xalan-2.5.

the *iTDS*, they were built based on a single level of granularity of data. For the *riTDS*, we designed two variants with the two filters (*riTDS-1* and *riTDS-2*) to simplify the set *rTDS*.

Table 4: Description of the metrics included in the data sets.

Variable	Description
CK suite (6)	
WMC	Weighted Methods per Class
DIT	Depth of Inheritance Tree
LCOM	Lack of Cohesion in Methods
RFC	Response for a Class
CBO	Coupling between Object classes
NOC	Number of Children
Martins metric (2)	
CA	Afferent Couplings
CE	Efferent Couplings
QMOOM suite (5)	
DAM	Data Access Metric
NPM	Number of Public Methods
MFA	Measure of Functional Abstraction
CAM	Cohesion Among Methods
MOA	Measure Of Aggregation
Extended CK suite (4)	
IC	Inheritance Coupling
CBM	Coupling Between Methods
AMC	Average Method Complexity
LCOM3	Normalized version of LCOM
McCabe's CC (2)	
MAX_CC	Maximum values of methods in the same class
AVG_CC	Mean values of methods in the same class
LOC	Lines Of Code
Bug	non-buggy or buggy

Second, we applied five typical classifiers for building defect predictors and compared their impacts on prediction results of the three types of TDS simplification methods in terms of evaluation measures.

Third, on the basis of the filtering strategies, we further sought the decision rule to determine an appropriate filter for

a given data set and tested its effectiveness compared with the results of the above methods with a single type of filter.

#### 4.3. Classifiers

In this study, prediction models were built with five well-known classification algorithms—namely, J48, Logistic Regression (LR), Naïve Bayes (NB), Support Vector Machine (SVM) and Random Forest (RF)—used in prior studies. All classifiers were implemented in Weka<sup>5</sup>. For our experiments, we used the default parameter settings for different classifiers specified in Weka unless otherwise specified.

**J48** is an open source Java implementation of the C4.5 decision tree algorithm in Weka, which is an extension of the ID3 algorithm and uses a divide and conquer approach to growing decision trees. For each variable  $X = \{x_1, x_2, \dots, x_n\}$  and the corresponding class  $Y = \{y_1, y_2, \dots, y_m\}$ , the information entropy and information gain are calculated as follows (Bhargava et al., 2013):

$$Entropy(X) = - \sum_{j=1}^n P(x_i) \log P(x_i), \quad (1)$$

$$Entropy(X|Y) = \sum_{i,j} P(x_i, y_j) \log \frac{P(y_j)}{P(x_i, y_j)}, \quad (2)$$

$$Gain(X, Y) = Entropy(X) - Entropy(X|Y), \quad (3)$$

where  $P(x_i)$  is the probability that  $X = x_i$ , and  $P(x_i, y_j)$  is the probability that  $X = x_i$  and  $Y = y_j$ .

**Naïve Bayes (NB)** is one of the simplest classifiers based on conditional probability, and it is termed as “naïve” because it assumes that features are independent, that is,  $P(\mathbf{X}|Y) = \prod_{i=1}^n P(X_i|Y)$ , where  $\mathbf{X} = (X_1, \dots, X_n)$  is a feature vector and  $Y$  is a class. Although the independence assumption is often violated in the real-world, Naïve Bayes has been proven

<sup>5</sup><http://www.cs.waikato.ac.nz/ml/weka/>



to be effective in many practical applications (Rish, 2001). A prediction model constructed by this classifier is a set of probabilities. Given a new class, the classifier estimates the probability that the class is buggy, based on the product of the individual conditional probabilities for the feature values in the class. Equation (4) is the fundamental equation for the Naïve Bayes classifier.

$$P(Y = k|\mathbf{X}) = \frac{P(Y = k) \prod_i P(X_i|Y = k)}{\sum_j P(Y = j) \prod_i P(X_i|Y = j)}. \quad (4)$$

**Logistic Regression (LR)** is used to learn functions of the form  $P(Y|\mathbf{X})$  in the case where  $Y$  is a discrete value and  $\mathbf{X} = (X_1, \dots, X_n)$  is any vector containing continuous or discrete values, and it directly estimates its parameters from training data. In this paper, we will primarily consider the case where  $Y$  is a binary variable (i.e., buggy or non-buggy). Note that the sum of equation (5) and equation (6) must equal 1, and  $w$  is the weight (Rish, 2001).

$$P(Y = 1|\mathbf{X}) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}. \quad (5)$$

and

$$P(Y = 0|\mathbf{X}) = \frac{\exp(w_0 + \sum_{i=1}^n w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}. \quad (6)$$

**Support Vector Machine (SVM)** is typically used for classification and regression analysis by finding the optimal hyperplane that maximally separates samples in two different classes. To classify  $m$  instances in the  $n$ -dimensional real space  $R^n$ , the standard linear SVM is usually used. A prior study conducted by Lessmann et al. (2008) showed that the SVM classifier performed as well as the Naïve Bayes classifier in the context of defect prediction.

**Random Forest (RF)** is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest (Breiman, 2001). In other words, RF is a collection of trees, where each tree is grown from a bootstrap sample. Additionally, the attributes used to find the best split at each node are a randomly chosen subset of the total number of attributes. Each tree in the collection is used to classify a new instance. The forest then selects a classification by choosing the majority result.

#### 4.4. Evaluation Measures

A binary classifier can make two possible errors: *false positive (FP)* and *false negative (FN)*. A correctly classified defective class is a *true positive (TP)* and a correctly classified non-defective class is a *true negative (TN)*. The prediction performance measures used in our experiments are described as follows:

- *Precision (prec)* addresses how many of the defective instances returned by a model are actually defective. The higher the precision is, the fewer false positives exist.

$$prec = \frac{TP}{TP + FP}. \quad (7)$$

- *Recall (pd)* addresses how many of the defective instances are actually returned by a model. The higher the recall is, the fewer false negatives exist.

$$pd = \frac{TP}{TP + FN}. \quad (8)$$

- *pf* (probability of false alarm) measures how many of the instances that triggered the predictor actually did not contain any defects. The best *pf* value is 0.

$$pf = \frac{FP}{FP + TN}. \quad (9)$$

- *f-measure* can be interpreted as a weighted average of *Precision* and *Recall*. The value of *f-measure* ranges between 0 and 1.

$$f - measure = \frac{2 * pd * prec}{pd + prec}. \quad (10)$$

- *g-measure* (the harmonic mean of *pd* and  $1 - pf$ ):  $1 - pf$  represents *Specificity* (the proportion of correctly identified defect-free instances) and is used together with *pd* to form the *G-mean<sub>2</sub>* measure. In our paper, we use these to form the *g-measure* as defined in (Peters et al., 2013).

$$g - measure = \frac{2 * pd(1 - pf)}{pd + (1 - pf)}. \quad (11)$$

- *Accuracy (acc)* measures how well a binary classification correctly identifies. The higher the accuracy is, the fewer errors made by a classifier exist. In this paper, it is used to measure the proportion of true recommendation results when answering the *RQ3* in the following section.

$$acc = \frac{TP + TN}{TP + FP + TN + FN}. \quad (12)$$

- *AUC* (the area under the Receiver Operating Characteristic (ROC) curve) is the portion of the area of unit square, equal to the probability that a classifier will identify a randomly chosen defective class higher than a randomly chosen defect-free one (Fawcett, 2006). An AUC value less than 0.5 indicates a very low true positive rate and high false alarm. As we know, compared with traditional accuracy measures, AUC is more suitable to reflect the performance of predictors regarding the problem of class distribution imbalance. Therefore, we also use AUC to evaluate the most suitable classifier for our method in *RQ2*.

In fact, the difference between the training set-driven filter and the test set-driven filter is determined by which data set (TDS or test) contains more information about defects (Peters et al., 2013). To reflect the comparison of defect information between TDS and the test set, the concept of defect proneness ratio (*DPR*) is introduced in our experiments. *DPR* represents the ratio of the proportion of defects in the training set to the proportion of defects in the test set. Intuitively, when the value



Table 5: The results of TDS simplification at different levels of granularity. The numbers in bold are the maximum among the five classifiers for each TDS simplification method in each scenario ( $r = 1, 2, 3$ ).

Strategies	Classifiers	f-measure			g-measure			#instances(simplified TDS)		
		1	2	3	1	2	3	1	2	3
<i>rTDS</i>	J48	0.334	0.348	0.336	0.402	0.425	0.425	387.4	798.1	1222.7
	LR	0.322	0.342	0.336	0.385	0.427	0.416			
	NB	<b>0.435</b>	<b>0.459</b>	<b>0.459</b>	<b>0.552</b>	<b>0.592</b>	<b>0.594</b>			
	RF	0.305	0.316	0.299	0.354	0.404	0.390			
	SVM	0.287	0.313	0.322	0.313	0.361	0.388			
<i>riTDS-1</i>	J48	0.337	0.347	0.371	0.400	0.430	0.466	316.7	537.9	722.9
	LR	0.334	0.365	0.362	0.393	0.445	0.448			
	NB	<b>0.437</b>	<b>0.461</b>	<b>0.465</b>	<b>0.565</b>	<b>0.595</b>	<b>0.606</b>			
	RF	0.327	0.308	0.309	0.388	0.392	0.405			
	SVM	0.292	0.320	0.319	0.315	0.368	0.385			
<i>riTDS-2</i>	J48	0.325	0.307	0.340	0.383	0.396	0.429	218.7	286.7	317.7
	LR	0.344	0.359	0.369	0.417	0.448	0.452			
	NB	<b>0.453</b>	<b>0.464</b>	<b>0.475</b>	<b>0.585</b>	<b>0.599</b>	<b>0.613</b>			
	RF	0.311	0.315	0.327	0.361	0.401	0.422			
	SVM	0.287	0.312	0.310	0.306	0.365	0.381			
<i>iTDS</i>	J48	0.340	0.338	0.343	0.469	0.440	0.467	209.8	503.4	697.2
	LR	0.357	0.346	0.338	0.477	0.450	0.442			
	NB	<b>0.466</b>	<b>0.460</b>	<b>0.458</b>	<b>0.611</b>	<b>0.610</b>	<b>0.610</b>			
	RF	0.336	0.319	0.324	0.452	0.427	0.441			
	SVM	0.310	0.300	0.305	0.394	0.381	0.389			

is approximately one, the relative proportions of defects in TDS and in the test set reach equilibrium.

$$DPR = \frac{\%Defects(trainingset)}{\%Defects(testset)}. \quad (13)$$

#### 4.5. Results

We organize our results according to the three research questions proposed in Section 1.

*RQ1: Does our TDS simplification method perform well compared with the benchmark methods?*

Given the strategies for TDS simplification at different levels of granularity, Table 5 shows some interesting results. First, the fine-grained strategy (*iTDS*) outperforms the coarse-grained strategy (*rTDS*) as a whole, indicated by the greater mean values of evaluation measures, especially for the *g-measure*. For example, the *g-measure* mean values of the *rTDS* with Naïve Bayes are 0.552, 0.592 and 0.594, respectively, but they are 0.611, 0.610 and 0.610 for the *iTDS*, respectively. Second, the result of the *riTDS* is approximately on the borderline between the *rTDS* and the *iTDS*, as it is a combination of the two methods, whereas some *f-measure* mean values of the *riTDS* are even better for those prediction models built with Logistic Regression and Naïve Bayes. Third, three out of five predictors (i.e., those built with LR, NB and RF) present a better *f-measure* and *g-measure* mean values with the *riTDS-2*, in particular, when increasing the value of the parameter  $r$ . That is, the filter based on training set-driven filtering strategy may in general work better on the instance-level simplification. It is worthwhile to note that the value of the parameter  $k$  mentioned in Algorithm 1 (line 15 and 21) is set to 10 because the same assignment was used in the prior studies (Peters et al., 2013; Turhan et al., 2009).

Regarding the necessity of TDS simplification, we then investigated the size of the final simplified TDS actually used to train defect predictors. As shown in Table 5, the last three columns list the corresponding average number of instances in simplified TDS in each scenario. Although the effect of the *riTDS* method on prediction is not always distinct, it is more effective from the perspective of TDS simplification. More specifically, compared with the simplification at a single level of granularity, there is a several-fold decrease in the number of useless instances with an increase of  $r$ , especially for the *riTDS-2*. Furthermore, it is obvious from Table 5 that a large increase in TDS's size (e.g., from 317 to 1222) does not significantly improve prediction performance, and sometimes it is just the opposite. That is, to a certain extent, the quality rather than the quantity of training data is a crucial factor that affects the performance of CPDP. This is one of our primary motivations to simplify the training data in this study.

To further investigate the practicability of our TDS simplification method, we compared the performance of the *riTDS* with the *iTDS* from the viewpoint of statistically significant difference. Table 6 presents the results of the Wilcoxon signed-rank test based on the *null hypothesis* that the medians of the two methods are identical (i.e.,  $H_0 : \mu_1 = \mu_2$ ). Obviously, the results highlight that there is no significant difference between the *riTDS* and the *iTDS*, indicated by all of the  $p > 0.05$  cases for the five typical classifiers. In other words, this suggests that the *riTDS* method can achieve satisfactory performance under the premise of using fewer instances for training, compared with the benchmark method.

Moreover, the *riTDS* method with different filters can achieve better precision than the *iTDS* method. In Table 7, it is clear that the degree of precision improvement of the *riTDS-2* is

Table 6: A comparison between *riTDS* and *iTDS*. *riTDS*/*iTDS* represents the ratio of the mean of the former to that of the latter, and *riTDS* vs. *iTDS* means the Wilcoxon signed-rank test of the distribution of prediction results of the two methods in terms of *f-measure* and *g-measure*.

Methods		<i>f-measure</i>						<i>g-measure</i>					
		<i>riTDS</i> / <i>iTDS</i>			vs. <i>iTDS</i> ( <i>Sig.p</i> = 0.01)			<i>riTDS</i> / <i>iTDS</i>			vs. <i>iTDS</i> ( <i>Sig.p</i> = 0.01)		
		1	2	3	1	2	3	1	2	3	1	2	3
<i>riTDS-1</i>	J48	0.991	1.029	1.084	0.700	0.884	0.270	0.852	0.978	0.997	0.228	0.980	0.739
	LR	0.936	1.055	1.071	0.871	0.489	0.469	0.824	0.989	1.014	0.158	0.858	0.782
	NB	0.937	1.002	1.015	0.086	0.765	0.549	0.924	0.974	0.994	0.012	0.437	0.993
	RF	0.974	0.965	0.953	0.791	0.437	0.782	0.858	0.918	0.917	0.096	0.164	0.544
	SVM	0.943	1.067	1.046	0.533	0.871	0.844	0.801	0.965	0.988	0.144	0.752	0.966
<i>riTDS-2</i>	J48	0.954	0.910	0.994	0.980	0.626	0.858	0.817	0.899	0.919	0.139	0.533	0.578
	LR	0.964	1.039	1.092	0.859	0.544	0.369	0.874	0.997	1.023	0.489	0.651	0.688
	NB	0.973	1.008	1.037	0.343	0.858	0.285	0.956	0.981	1.006	0.203	0.293	0.437
	RF	0.998	0.989	1.006	0.457	0.884	0.726	0.799	0.938	0.958	0.027	0.285	0.925
	SVM	0.925	1.041	1.016	0.427	0.966	0.912	0.776	0.956	0.977	0.080	0.925	0.993

Table 7: A comparison of the precision of the *riTDS* and the *iTDS*, and  $\Delta$  represents the relative increment of precision.

Methods		precision			$\Delta$ ( <i>riTDS</i> - <i>iTDS</i> )		
		1	2	3	1	2	3
<i>riTDS-1</i>	J48	0.437	0.403	0.426	0.001	-0.054	0.024
	LR	0.435	0.432	0.413	0.044	0.054	0.066
	NB	0.550	0.556	0.560	0.030	0.030	0.066
	RF	0.398	0.334	0.345	0.003	0.003	0.040
	SVM	0.391	0.366	0.337	-0.002	0.004	0.009
<i>riTDS-2</i>	J48	0.427	0.360	0.405	0.066	0.047	0.051
	LR	0.472	0.449	0.429	0.110	0.123	0.106
	NB	0.576	0.584	0.612	-0.002	0.028	0.057
	RF	0.381	0.352	0.358	0.058	0.050	0.043
	SVM	0.380	0.358	0.347	0.108	0.094	0.077

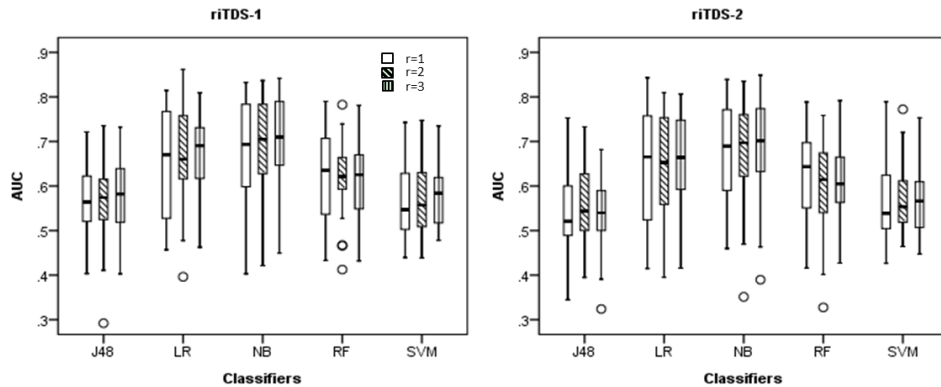


Figure 6: The standardized boxplots of the distributions of AUC values based on the *riTDS-1* and the *riTDS-2*. From the bottom to the top of a standardized box plot: minimum, first quartile, median, third quartile and maximum. The outliers are plotted as circles.

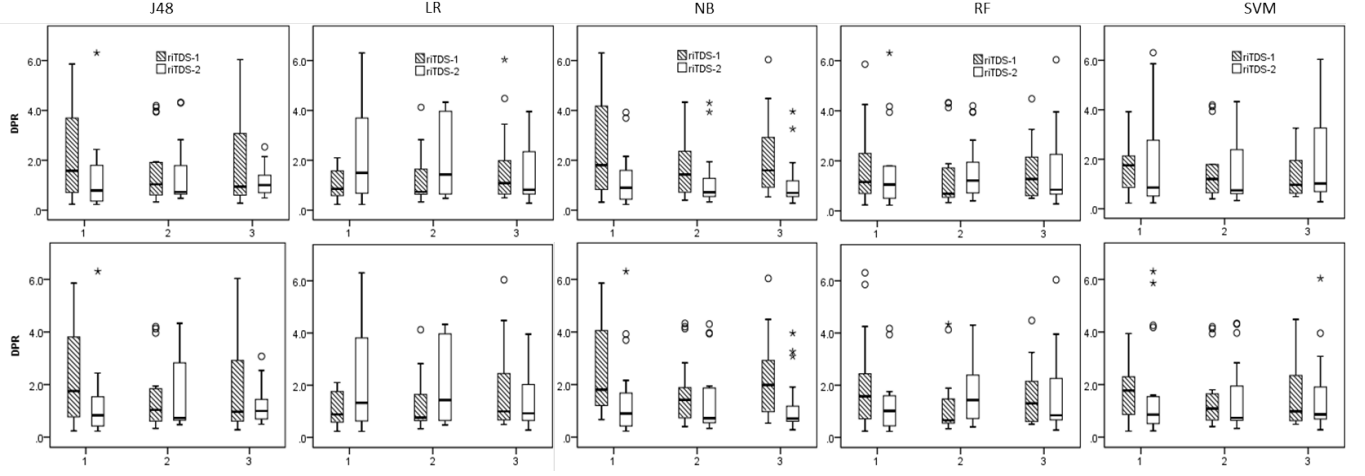


Figure 7: The standardized boxplots of the DPR distribution of predictions in the groups riTDS-1 and riTDS-2 using *f-measure* (up) and *g-measure* (down) as the group division standard. From the bottom to the top of a standardized box plot: minimum, first quartile, median, third quartile and maximum. The outliers are plotted as circles and pentagrams.

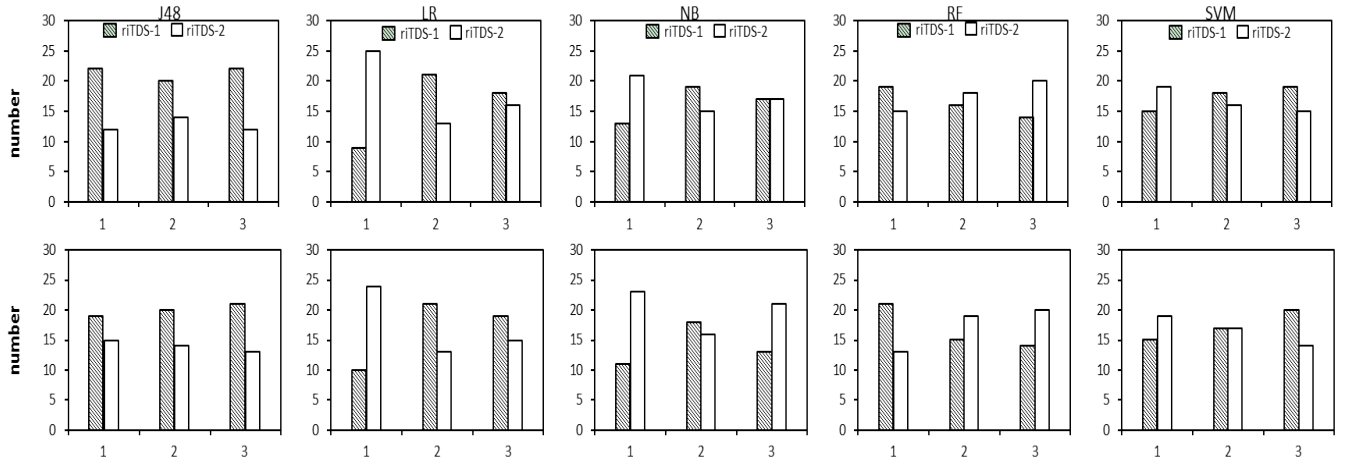


Figure 8: The comparison between the groups riTDS-1 and riTDS-2 using *f-measure* (up) and *g-measure* (down) as the group division standard. The number of elements in the groups is counted among the 34 CPDP cases.

greater than that of the *riTDS-1*, and the results of the *riTDS-2* with LR and SVM are more significant. Therefore, our TDS simplification method not only achieved a comparative *f-measure* and *g-measure* values, but also significantly reduced the number of training instances and improved the performance in terms of precision.

*RQ2: Which classifier is more suitable for CPDP with our TDS simplification method?*

The numbers in bold in Table 5 indicate that the predictor built with Naïve Bayes yields the best performance because of the greatest *f-measure* and *g-measure* mean values, followed by those built with Logistic Regression and J48. With regard to AUC value, Figure 6 further validates that Naïve Bayes is the best classifier and that Logistic Regression is an alternative in our context. However, J48 presents an obvious disadvantage because of its lower median AUC value, although it shows middling performance in terms of *f-measure* and *g-measure* mean values.

Interestingly, whichever level of granularity we select, the predictor built with SVM seems to have the worst performance, especially when using the *rTDS* method. Our results also validate the statement that simple learning algorithms tend to perform well for defect prediction (Hall et al., 2012). In the literature (Herbold, 2013), the author weighted the training instances, thus leading to a remarkable performance improvement by the SVM classifier. The reason why we did not take the weight of training data into account is that we focused primarily on understanding the differences between TDS simplification methods from the perspective of granularity (e.g., release-level vs. instance-level). Hence, we used the same data processing method for all classifiers under discussion, without considering specific optimization for any one of the classifiers.

In addition, for each scenario ( $r = 1, 2, 3$ ), we divided the 34 CPDP cases into two groups according to their performance measures (*f-measure* and *g-measure*). That is, for the  $i^{th}$  target release, if  $measure_{ir}^{riTDS-1} > measure_{ir}^{riTDS-2}$ , this CPDP is classified into the group *riTDS-1*; otherwise, it belongs to the group *riTDS-2*. We then compared the distribution of *DPR* values between the group *riTDS-1* and the group *riTDS-2* in terms of *f-measure* and *g-measure*. Figure 7 shows that for those predictions with Naïve Bayes, the group *riTDS-1* has a significantly higher median *DPR* value than the group *riTDS-2*, and this trend is independent of the parameter  $r$ . Specifically, the median *DPR* values of the former are more than twice those of the latter. For example, the median *DPR* values of the two groups are 1.59 vs. 0.691 (*f-measure*) and 1.98 vs. 0.706 (*g-measure*), respectively, when returning the top three releases as the set *rTDS*. In addition, J48 and SVM show a similar trend except in the scenario  $r = 3$ . The obvious difference in *DPR* values of CPDP is a meaningful insight into how to determine an appropriate filter for instance simplification in the *riTDS*. Therefore, the predictor built with Naïve Bayes is still the most suitable prediction model due to its ability to distinguish different filters. The discussion on filter selection in terms of *DPR* will be introduced in the following subsection.

*RQ3: Which filter for TDS simplification should be*

*preferable in a specific scenario?*

For different scenarios about  $r$ , on the basis of the aforementioned groups, Figure 8 shows the number of elements in each group. Although the results of LR and SVM are similar to each other, there are no universal patterns for all classifiers. The results indicate that some CPDP cases are indeed preferable to the *riTDS-1*, while others are yet apt to use the *riTDS-2*. For example, for all scenarios with J48, the group *riTDS-1* has higher bars than the group *riTDS-2* using both *f-measure* and *g-measure* as the group division standard, which, in turn, has more elements when using Random Forest except in the scenario  $r = 1$ . Thus, it is very clear that the above findings drawn from Figure 7 and Figure 8 only show an overall difference between the two filters for instance simplification in CPDP (*riTDS-1* and *riTDS-2*), but they cannot yet effectively help us make a reasonable decision on the choice of an appropriate filtering strategy.

To solve this problem, we first gathered the 102 ( $3 \times 34 = 102$ ) predictions used in our experiments, and then divided them into two groups according to the similar rule mentioned above. The groups *riTDS-1* and *riTDS-2* will be viewed as the actual observations in the following tasks. According to the *DPR* distribution in Figure 7, we suppose that the *riTDS-1* filter is recommended to a target release if its *DPR* value is not less than  $\rho$ ; otherwise, the *riTDS-2* filter is recommended. This assumption is named as  $\rho+$ . Thus, the value of *accuracy* is calculated using the Eq. (12), where *TP* and *TN* represent the correct recommendation for the groups *riTDS-1* and *riTDS-2* with a specific  $\rho$ , respectively. Note that,  $\rho \in [min, max]$ , where *min* and *max* are the minimum and maximum *DPR* values among the 102 predictions. The higher the *accuracy* value is, the more reliable the choice of filters made by  $\rho$ . Figure 9 shows that the *accuracy* values reach a peak when  $\rho$  changes from the minimum to the maximum. With the optimal *accuracy* value, it is not hard to make a choice between the *riTDS-1* filter and the *riTDS-2* filter when using a specific classifier. That is, we can employ the parameter  $\rho$  as a corresponding threshold to determine the eventual choice of filtering strategies. Interestingly, each classifier has the same optimal  $\rho$  value using whichever measure as the group division standard. For example, with respect to Naïve Bayes, the *riTDS-2* filter should be recommended if the *DPR* value of a target release is 1.0; otherwise, the *riTDS-1* filter should be preferable if the value equals 1.5.

To further identify the appropriate threshold of the  $\rho$  value for each classifier, we conducted another experiment with the opposite assumption (named as  $\rho-$ ). That is, the *riTDS-2* filter is recommended to a target release if its *DPR* value is not less than  $\rho$ ; otherwise, the *riTDS-1* filter is recommended. In Figure 10, the overall optimal *accuracy* values of four cases declined, in particular for the case of Naïve Bayes where the maximum values are only 0.52 and 0.59 when using *f-measure* and *g-measure* as the group division standard, respectively. In fact, these two results indicate the case in which all predictions used the *riTDS-2* filter because 0.23 is the lowest *DPR* value. However, Logistic Regression achieves a higher *accuracy* and larger optimal  $\rho$  value according to

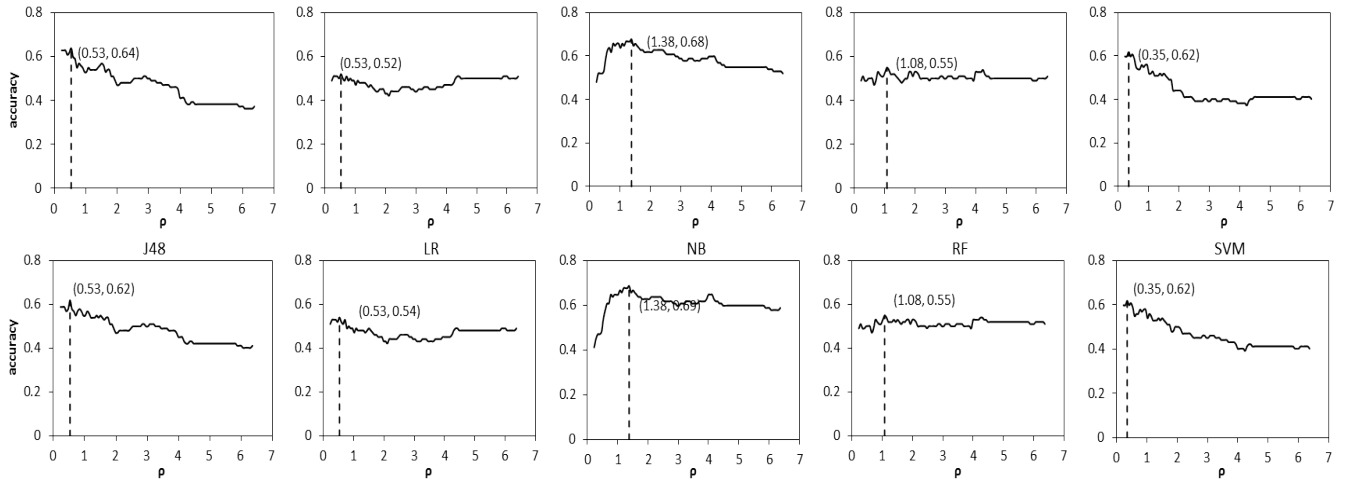


Figure 9: The recommendation *accuracy* value changes with the threshold  $\rho$  of DPR values according to the assumption that the *riTDS-1* filter is recommended if  $DPR \geq \rho$  (named as  $\rho+$ ). The groups in Figure 8 are viewed as the actual results in our experiment.

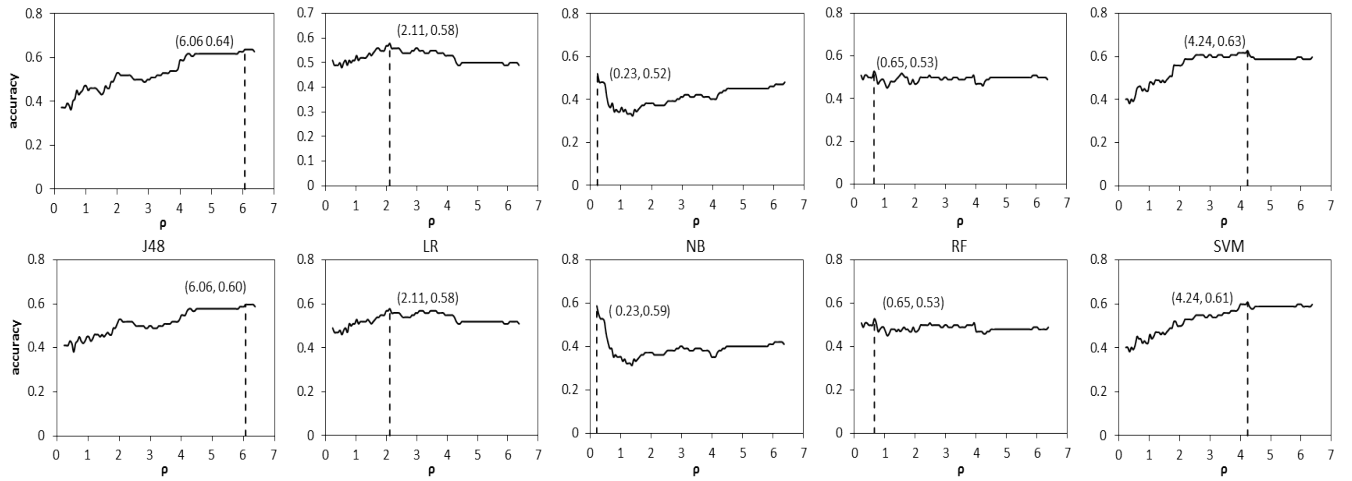


Figure 10: The recommendation *accuracy* value changes with the threshold  $\rho$  of DPR values according to the opposite assumption that the *riTDS-2* filter is recommended if  $DPR \geq \rho$  (named as  $\rho-$ ). The groups in Figure 8 are viewed as the actual results in our experiment.

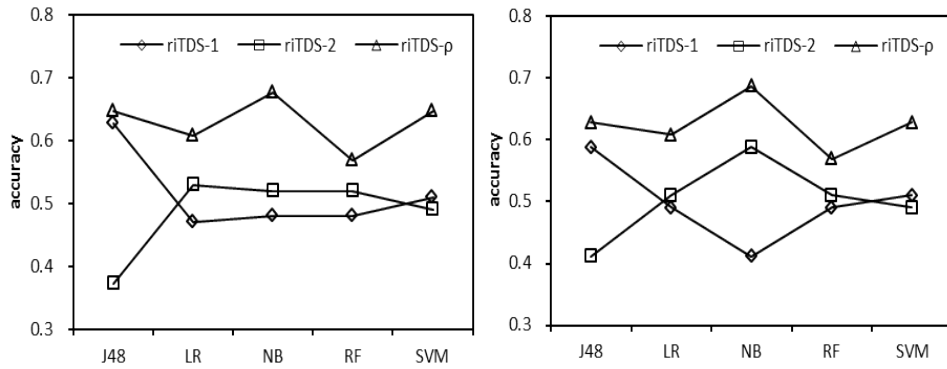


Figure 11: The comparison of recommendation *accuracy* among the predictors built with different filters using *f-measure* (left) and *g-measure* (right) as the group division standard.

Table 8: The threshold of  $\rho$  (the range of DPR) for the *riTDS-1* filter.

Classifier	Range
J48	$0.53 \leq DPR < 6.06$
LR	$0.53 \leq DPR < 2.11$
NB	$1.38 \leq DPR$ or $DPR < 0.23$
RF	$1.08 \leq DPR$ or $DPR < 0.65$
SVM	$0.35 \leq DPR < 4.24$

Table 9: The comparison between different filters with regard to recommendation *accuracy*.

Grouping	riTDS	J48	LR	NB	RF	SVM
<i>f-measure</i>	$-\rho$	0.647	0.608	0.676	0.569	0.647
	-1 (%)	+3.1	+29.2	+40.8	+18.4	+26.9
	-2 (%)	+73.7	+14.8	+30.2	+9.4	+32.0
<i>g-measure</i>	$-\rho$	0.627	0.608	0.686	0.569	0.627
	-1 (%)	+6.7	+24.0	+66.7	+16.0	+23.1
	-2 (%)	+52.4	+19.2	+16.7	+11.5	+28.0

the opposite assumption, which is consistent with the *DPR* distribution shown in Figure 7. According to the results of Figure 9 and Figure 10, the threshold of  $\rho$  used to determine the *riTDS-1* filter for each classifier can be identified in Table 8, whereas the corresponding complementary set is suitable for the *riTDS-2* filter.

With the threshold of  $\rho$ , we compared the recommendation *accuracy* among the three cases with different filters: the *riTDS-1* filter, the *riTDS-2* filter, and filter selection determined by the *DPR* value. The results show that our approach increases the *accuracy* value, in particular for Logistical Regression, Naïve Bayes and SVM (see Figure 11). For example, compared with the filters *riTDS-1* and *riTDS-2*, for Naïve Bayes, the *riTDS* -  $\rho$  filter achieves a marked increase in *accuracy* when using the *f-measure* and *g-measure* as the group division standard. The values in terms of different groupings grow by 40.8% and 30.2%, and 66.7% and 16.7%, respectively (see Table 9). The improvement of recommendation *accuracy* indicates that our approach to determining the appropriate filter for TDS simplification is feasible and outperforms the *riTDS* with a single type of filter.

To further validate the feasibility of our approach, we compared the prediction performance among the three cases in terms of *f-measure* and *g-measure*. Figure 12 shows that our approach achieves various degrees of improvement in the *f-measure* and *g-measure* values overall, in particular for the prediction built with Logistical Regression and Naïve Bayes when  $r$  is 2 and 3. Note that, the improvement is optimistic compared with the best case where the filter with the greater measure value is applied to a target release, although the degree of improvement does not seem to be great. In addition, we also compared the prediction *precision* of our approach with the two benchmark filters. Again, there is an overall growth trend for the five classifiers (see Table 10). This evidence suggests that our approach is also feasible in terms of prediction performance.

Table 10: The increment of prediction *precision* for *riTDS* -  $\rho$ .

Classifier	riTDS	1	2	3
J48	$-\rho$	0.437	0.395	0.428
	-1	0.000	-0.008	+0.002
	-2	+0.010	+0.035	+0.023
LR	$-\rho$	0.446	0.444	0.420
	-1	+0.011	+0.012	+0.007
	-2	-0.026	-0.005	-0.009
NB	$-\rho$	0.577	0.575	0.612
	-1	+0.027	+0.019	+0.052
	-2	+0.001	-0.009	0.000
RF	$-\rho$	0.396	0.339	0.364
	-1	-0.002	+0.005	+0.019
	-2	+0.015	-0.013	+0.006
SVM	$-\rho$	0.391	0.357	0.348
	-1	0.000	-0.009	+0.011
	-2	+0.011	-0.001	+0.001

## 5. Discussion

*RQ1:* A larger amount of training data may not lead to a higher performance of CPDP, suggesting the necessity of simplifying training data. However, none of the existing methods take the levels of granularity of data into consideration, especially with regard to multiple granularity (e.g., the two-step strategy proposed in our paper), which is a key factor for building practical CPDP models. As we consider such a factor, our experimental results show that less instances are involved in training the predictors based on multiple levels of granularity compared with those based on a single level of granularity, with little loss of accuracy. The simplified TDS preserves the most relevant training instances, which is helpful to reduce the false alarms and build the quality predictors.

The prediction results of different predictors based on the methods *rTDS*, *iTDS* and *riTDS* were calculated without any feature selection techniques. That is, for the simplified TDS, all predictors were built with the twenty software metrics (viz. features). As shown in Figure 1, this paper focuses on how to reduce data volumes in a TDS. If we applied feature selection techniques to building defect predictors, it is hard to distinguish what factor actually obtained the greater improvement on prediction performance. Therefore, we did not consider feature selection in our experiments. Additionally, the parameter  $r$  was set to no more than 3 because 8 out of 10 projects under discussion have no more than 4 releases available. That is, the majority of projects have to select no more than 3 releases as training data even if we conduct experiments on WPDP. Prior studies (He et al., 2012) have also used the same setting for the parameter  $r$ .

*RQ2:* As we know, Naïve Bayes has been validated as a robust machine learning algorithm for supervised software defect prediction problems in both WPDP and CPDP. Interestingly, our result is completely consistent with the conclusions drawn in the literature (Hall et al., 2012; Catal, 2011), that is, Naïve Bayes outperforms the other typical classifiers within our CPDP context in terms of *f-measure*, *g-measure* and AUC. It is worthwhile to note that different

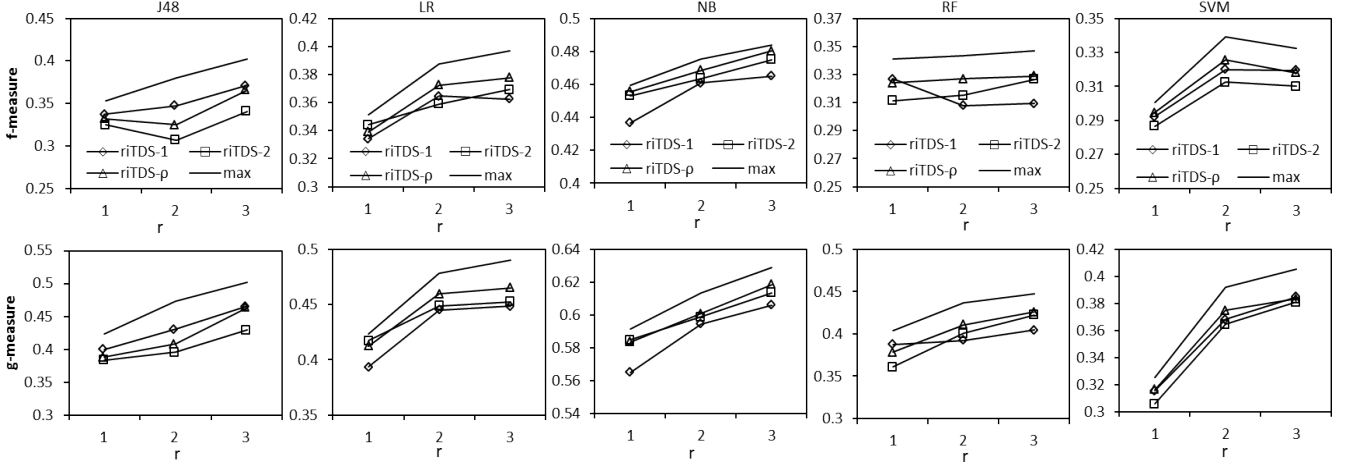


Figure 12: The improvement of  $f$ -measure and  $g$ -measure for  $riTDS - \rho$ .

prediction models were built based on these classifiers without specific optimization because in this study, we focused primarily on the levels of granularity and filtering strategies for TDS simplification. However, the performance differences between different prediction models indicate that simple classifiers, such as Naïve Bayes, can be preferable to training data of quality.

In addition, for  $DPR$ , different classifiers exhibit different abilities to distinguish the results of the two filters in question. For example, the group  $riTDS-1$  has a higher median  $DPR$  value than the group  $riTDS-2$  except Logistical Regression when the parameter  $r$  is 1. However, the opposite results occur using Logistical Regression and Random Forest when  $r$  is 2. J48 and SVM have the approximate median  $DPR$  values between the group  $riTDS-1$  and the group  $riTDS-2$  when  $r$  is 3, although they maintain a similar trend in the first two  $r$  values. However, the best ability of Naïve Bayes to distinguish the group  $riTDS-1$  and the group  $riTDS-2$  paves a way for the feasibility and generality of our approach proposed to answer  $RQ3$ .

**$RQ3$ :** As an alternative strategy, the training set-driven filter for TDS simplification is in general better than the test set-driven filter, which is consistent with the findings obtained in (Peters et al., 2013). However, the authors did not analyze the specific application scenarios for each type of filter. We filled the gap in terms of recommendation *accuracy* based on  $DPR$  value, and found that the training set-driven filter is more suitable for those predictions with very low or very large  $DPR$  values when using J48, LR, and SVM classifiers. Conversely, a prediction with a middle  $DPR$  value is more likely to choose the training set-driven filter when using NB and RF classifiers. Note that, to make the right decision between the training set-driven filter and the test set-driven filter according to the value of  $DPR$ , we seek the optimal point  $\rho$  through gradually changing the value of  $DPR$  with an increment of  $\frac{max-min}{100}$ .

With regard to the threshold of  $\rho$ , we have to admit that we may obtain different thresholds for such an index if other formulas are used to evaluate the recommendation results. Nevertheless, we still obtained various valuable findings. For

example, the test set-driven filter is preferable when the  $DPR$  value is between 1.38 and 2.11, and this range is suitable for all five of the classifiers in our context. Although there are no common ranges for training set-driven filter selection, our results still indicate that the practical guideline for the decision-making on which filtering strategy is suitable for instance selection indeed exists, and that it does improve the prediction performance of those predictors based on a single type of filter.

## 6. Threats to Validity

In this study, although we obtained several interesting findings according to the three research questions proposed in Section 1, some potential threats to the validity of our work still exist.

Threats to *construct validity* are primarily related to the data sets we used. All of the data sets were collected by Jureczko and Madeyski (Jureczko and Madeyski, 2010) and Jureczko and Spinellis (Jureczko and Spinellis, 2010) with the support of existing tools: BugInfo and Ckjm. These data sets have been validated and applied to several prior studies, though errors in the process of defect identification may exist. Therefore, we believe that our results are credible and can be reproduced. Additionally, we applied a log transformation to feature values before building defect predictors, and we cannot ensure that it is better than other preprocessing methods. The impact of data preprocessing on prediction performance is also an interesting problem that needs further investigation.

Threats to *internal validity* are mainly related to various learning algorithm settings in our study. For our experiments, although the  $k$ -nearest neighbors algorithm (KNN) was selected as the basic selection algorithm, we are aware that our results would change if we were to use a different method. However, to the best of our knowledge, both KNN and its variants were successfully applied to TDS simplification in several prior studies (Peters et al., 2013; Herbold, 2013). Moreover, we did not implement specific optimization for any classifiers in question when building different prediction models because the



goal of this experiment is not to improve the performance of a given classifier.

Threats to *external validity* could be related to the generality of the results to other on-line public data sets used for defect prediction, such as NASA and Mozilla. The data sets used in our experiments are chosen from a small subset of all projects in the PROMISE repository, and it is possible that we accidentally selected data sets that have better (or worse) than average CPDP performance, implying that some of our findings (e.g., the threshold of  $\rho$  for the five typical classifiers) might not be generalizable to other data sets.

## 7. Conclusion

TDS simplification, which filters out the irrelevant and redundant training data, plays an important role in building better CPDP models. This study reports an empirical study aiming at investigating the impact of the level of granularity and filtering strategy on TDS simplification. The study has been conducted on 34 releases of 10 open-source projects in the PROMISE repository and consists of (1) a comparison between multi-granularity and benchmark (single level of granularity) TDS simplification, (2) a selection of the best classifier in our context, and (3) an assessment of practical selection rules for the state-of-the-art filtering strategies for instance simplification.

The results indicate that the CPDP predictions based on the multi-granularity simplification approach (e.g., the two-step strategy proposed in our paper) capture competitive *f-measure* and *g-measure* values showing no statistically significant differences compared with those benchmark TDS simplification approaches, and that the size of simplified TDS was sharply reduced with an increase in the number of returned neighbors at the level of release. In addition, our results also show that more actually defective instances can be predicted by our method and that Naïve Bayes is more suitable for building predictors for CPDP with simplified TDS. Finally, the *DPR* index is useful in determining a proper filtering strategy when using the *riTDS* method, and the practical selection rule based on the *DPR* value does improve prediction performance to some extent.

Our future work will focus mainly on two aspects: on the one hand, we will collect more open-source projects (e.g., Eclipse and Mozilla) to validate the generality of our approach; on the other hand, we will further consider the number of defects of an instance to provide an effective TDS simplification method for CPDP.

## Acknowledgment

This work is supported by the National Basic Research Program of China (No. 2014CB340401), the National Natural Science Foundation of China (Nos. 61273216, 61272111, 61202048 and 61202032), the Science and Technology Innovation Program of Hubei Province (No. 2013AAA020), the National Science and Technology Pillar Program of China

(No. 2012BAH07B01), the open foundation of Hubei Provincial Key Laboratory of Intelligent Information Processing and Real-time Industrial System (No. znss2013B017), and the Youth Chenguang Project of Science and Technology of Wuhan City in China (No. 2014070404010232).

## References

- He Z., Shu F., Yang Y., *et al.*, An investigation on the feasibility of cross-project defect prediction, *Automated Software Engineering*, 2012, 19(2): 167-199.
- Zimmermann T., Nagappan N., Gall H., *et al.*, Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, In: *Proceedings of the 7th Joint Meeting of the European Software Engineering conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering*, 2009: 91-100.
- Jureczko M., Madeyski L., Towards identifying software project clusters with regard to defect prediction, In: *Proceedings of the 6th International Conference on Predictive Models in Software Engineering Timisoara, Romania*, 2010: 1-10.
- Jureczko M., Spinellis D., Using object-oriented design metrics to predict software defects, In: *Proceedings of the 15th International Conference on Dependability of Comp. System, Monographs of System Dependability*, 2010: 69-81.
- Peters F., Menzies T. and Marcus A., Better cross company defect prediction, In: *Proceedings of the 10th Workshop on Mining Software Repositories*, 2013: 409-418.
- Weyuker E.J., Ostrand T.J., Bell R.M., Comparing the effectiveness of several modeling methods for fault prediction, *Empirical Software Engineering* 2009, 15(3): 277-295.
- Tosun A., Bener A., Kale R., AI-based software defect predictors: applications and benefits in a case study, In: *Proceedings of the 22th Innovative Applications of Artificial Intelligence Conf.*, 2010: 1748-1755.
- D'Ambros M., Lanza M., Robbes R., An extensive comparison of bug prediction approaches, In: *Proceedings of the 7th Working Conference on Mining Software Repositories*, 2010: 31-41.
- Rahman F., Posnett D., Devanbu P., Recalling the imprecision of cross-project defect prediction, In: *Proceedings of the 20th International Symp. on the Foundations of Software Engineering* 2012: 61.
- Briand L. C., Melo W. L., Wst J., Assessing the applicability of fault-proneness models across object-oriented software projects, *IEEE Transactions Software Engineering* 2002, 28(7): 706-720.
- Turhan B., Menzies T., Bener A., *et al.*, On the relative value of cross-company and within-company data for defect prediction, *Empirical Software Engineering* 2009, 14(5): 540-578.
- Turhan B., Misirli A. T. and Bener A., Empirical evaluation of the effects of mixed project data on learning defect predictors, *Information and Software Technology*, 2013, 55(6): 1101-1118.
- Lu H., Cukic B. and Culp M., Software defect prediction using semi-supervised learning with dimension reduction, In: *Proceedings of the 27th International Conference on Automated Software Engineering* 2012: 314-317.
- Herbold S., Training data selection for cross-project defect prediction, In: *Proceedings of the 9th International Conference on Predictive Models in Software Engineering ACM*, 2013: 6.
- Hall T., Beecham S., Bowes D., *et al.*, A systematic review of fault prediction performance in software engineering, *IEEE Transactions on Software Engineering*, 2012, 38(6): 1276-1304.

- Catal C., Software fault prediction: A literature review and current trends, *Expert Systems with Applications*, 2011, 38(3): 4626-4636.
- Ericsson M., Löwe W., Olsson T. and Toll D., *et al.*, A Study of the Effect of Data Normalization on Software and Information Quality Assessment, *International Workshop on Quantitative Approaches to Software Quality*, 2013: 55-60.
- Lessmann S., Baesens B., Mues C., *et al.*, Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Transactions on Software Engineering* 34(2008) 485-496.
- Nam J., Pan S. J., Kim S., Transfer defect learning, In: *Proceedings of the 35th International Conference on Software Engineering*, San Francisco, CA, USA, 2013: 382-391.
- Bishop C. M. and Nasrabadi N. M., *Pattern recognition and machine learning*, New York: Springer, 2006.
- Bhargava N., Sharma G., Bhargava R., *et al.*, Decision Tree Analysis on J48 Algorithm for Data Mining, *International Journal of Advanced Research in Computer Science and Software Engineering*, 2013,3(6): 1114-1119.
- Rish I., An empirical study of the naive Bayes classifier, In: *Proceeding of the IJCAI 2001 the Workshop on Empirical Methods in Artificial Intelligence (IJCAI'01-EMPAI)*, Washington, USA, 2001, pp. 41-46.
- Breiman L., Random Forests, *Machine Learning*, 2001,45 (1): 5-32.
- Fawcett T., An introduction to ROC analysis, *Pattern Recognition Letters*, 2006,27(8): 861-874.
- Ma Y., Luo G., Zeng X., *et al.*, Transfer learning for cross-company software defect prediction, *Information and Software Technology*, 2012, 54(3): 248-256.
- Pan S. J., and Yang Q., A survey on transfer learning, *IEEE Transactions on Knowledge and Data Engineering* 2010, 22(10): 1345-1359.
- Kocaguneli E., Menzies T. and Mendes E., Transfer learning in effort estimation, *Empirical Software Engineering* 2014: 1-31.
- Menzies T., Butcher A, Cok D., *et al.*, Local versus Global Lessons for Defect Prediction and Effort Estimation, *IEEE Transactions on Software Engineering* 2013, 39(6): 822-834.
- Posnett D., Filkov V., Devanbu P., Ecological inference in empirical software engineering, In: *Proceedings of the 26th International Conference on Automated Software Engineering IEEE*, 2011: 362-371.
- Bettenburg N., Nagappan M., Hassan A E., Think locally, act globally: Improving defect and effort prediction models, In: *Proceedings of the 9th Working Conference on Mining Software Repositories, IEEE*, 2012: 60-69.
- Xue G. R., Dai W., Yang Q., *et al.*, Topic-bridged PLSA for cross-domain text classification, In: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008: 627-634.
- A. Arnold, R. Nallapati, W.W. Cohen, A comparative study of methods for transductive transfer learning, In: *Proceedings of the International Conference on Data Mining*, 2007: 77-82.
- Pan S. J., Ni X., Sun J. T., *et al.*, Cross-domain sentiment classification via spectral feature alignment, In: *Proceedings of the 19th International Conference on World Wide Web*, ACM, 2010: 751-760.
- Zhang X., Dai W., Xue G., *et al.*, Adaptive Email Spam Filtering based on Information Theory, In: *Proceedings of the 8th International Conference on Web Inform. Sys. Engineering* 2007: 59-170.
- He P., Li B., Liu X., *et al.*, An Empirical Study on Software Defect Prediction with a Simplified Metric Set, *arXiv:1402.3873*, 2014.
- Han J., Kamber M. and Pei J., *Data mining: concepts and techniques*, 3rd ed. Waltham, Mass.: Elsevier/Morgan Kaufmann, 2012.
- Kotsiantis S. B., Kanellopoulos D. and Pintelas P. E., Data preprocessing for supervised learning, *International Journal of Computer Science*, vol. 1, 2006.
- Menzies T., DiStefano J. S., Chapman M., and McGill K., Metrics that Matter, In: *Proceedings of the 27th NASA SEL Workshop Software Engineering* 2002.
- Rainer A. and Gale S., Evaluating the quality and quantity of data on open source software projects, In: *Proceedings of the 1st International Conference on Open Source Software*. 2005:29-36.